Amendments to the Specification

Please replace the paragraph beginning on page 3, line 23 with the following rewritten paragraph:

Current data transformation techniques are generally expensive to implement, are not portable, and difficult to adapt to new or changing circumstances. For example, point-to-point links are generally hand-coded customized data transformation programs. Customized code is typically written in-house and is specific to a single application or DBMS environment. On the positive side, such solutions generally provide exactly what is needed and no more, and address requirements for which there may be no off-the-shelf products. In-house development, testing and debugging also narrows the focus, and tends to produce a workable, if non-versatile, solution. On the other hand, because these routines are usually specific to a particular source or target database, they are difficult to port to other environments. These routines may also be difficult to repeat because the routines are generally unique to each situation and because there is typically no infrastructure in place to manage the processes. Finally, building custom routines robs in-house DBAs of time better spent on other tasks. In addition, custom coded solutions require continued maintenance because they must be modified every time a new requirement is added to the system Further, custom code may take a relatively long time to implement with some legacy migration projects tying up critical IT staff for weeks, ~~month~~ months and even years.

Please replace the paragraph beginning on page 15, line 5 with the following rewritten paragraph:

Map designer **210** provides an interface that permits user **10** to specify the relationships between data (called "mappings") retrieved from source **124** and written to target **134**. In some embodiments, this interface may allow the mappings to be established graphically. In other embodiments, the mappings may be specified as a set of one or more expressions or rules. In yet other embodiments, a default set of mappings may be used in the absence of the user input. In one particular embodiment, the default mapping is a one-to-one mapping between the source **124** and the target **134**.

Please replace the paragraph beginning on page 16, line 1 with the following rewritten paragraph:

When a transformation is executed, transformation engine **110** retrieves the applicable transformation rules from either ~~from~~ the map designer or the transformation map repository. Transformation engine **110** then directs source spoke **120** to retrieve the applicable data from source **124**. Transformation engine **110** performs the necessary functions on the data and then ~~send~~ sends the data to

target spoke **130** to be stored in target **134** as specified by the transformation rules in the transformation map.

Please replace the paragraph beginning on page 16, line 23 with the following rewritten paragraph:

ConnectionInfo objects **322** and **332** manage information used to connect to data source **124** and data target **134**, respectively. ConnectionInfo objects **322** and **332** may be used by source spoke **120** and target spoke **130** to provide access to source object **124** and target object **134**, respectively. For simple ~~filed-based~~ file-based data sources, this may just be the name of the file (local or remote file system of a URI). For database management systems, this information could most often include the name of the DBMS server, the name of the database, a user ID, a password, and a reference to the DBMS object (table name or a query result). The connection information may also include option settings to control the behavior of the spoke used for the data access.

Please replace the paragraph beginning on page 17, line 10 with the following rewritten paragraph:

EventHandler object **342** manages a collection of actions **350** used to handle a specific type of event. EventHandler object **342** is also responsible for compiling and executing actions **350**. EventHandlers **340** is an array of EventHandler objects **342** ~~objects~~. Five types of EventHandlers **340** are used to handle processing of transformation **300**, source **324**, target **334**, source record layout, and target record layout events.

Please replace the paragraph beginning on page 17, line 16 with the following rewritten paragraph:

Generally, the Actions **350** defined in the EventHandlers **340** are used to direct operations in the transformation engine. Permissible Actions **350** may be dictated by several factors, including the capabilities of the transformation engine, and the type of target spoke or adaptor that is used. In some embodiments, a "modal" target adaptor is used. In these embodiments, the type of operation performed by the target adaptor is implied by the current output mode. As described below in more detail, the output mode may be specified when defining the target. Examples of output modes include, without limitation: **Replace File/Table** – create a new data target or table, OR overwrite both the data and the structure of an existing file or table; **Append to File/Table** – keep the existing records and add new records to an existing file or table, OR add records to an empty table; **Update File/Table** – searches an existing data target for a match in the key fields, which the user defines in target keys/indexes, and updates data in the specified

manner; **Clear File/Table c ntents and Append** – preserves the target record layout and the relationships between tables (if the user has defined them) and discards any existing records; **Delete from File/Table** – searches an existing data target for a match in the key fields, which the user has defines in target keys/indexes, and deletes data in the specified manner. In some embodiments of the present invention, event actions that may be used with modal target adaptors include, without limitation:

- **Abort** – used to abort the execution of the execution
- **Resume** – used within an error event handler to resume execution of an action list which was interrupted by an exception
- **Clear** – used to clear the values for a target record layout
- **Map** – execute the mapping expressions for a target record layout
- **Put** – write a record to a target
- **ClearMapPut** – a composition action combining the clear, map, and put actions
- **Execute** – used to execute expression language code
- **LogMessage** – used to write a message to the transformation log file
- **LogTargetRecord** – used to display the contents of a target record in the transformation log
- **ClearInitialize** – used to initialize a target record with default values

It is noted that these event actions may perform different functions depending on the active output mode. For example, in a **Replace File/Table** output mode, the **Put** command would essentially erase an existing record in the target and write a new record in its place. In contrast, during ~~a~~an **Append to File/Table** output mode, the **Put** command would append a new record to the target. In these embodiments, the specification of the target object and the operation on that object may be specified by the target adaptor.

Please replace the paragraph beginning on page 17, line ~~16~~ 22 with the following rewritten paragraph:

In other embodiments, the specification of the target object and the operation on an object may be specified in the Actions **350** used in the event handlers for the transformation engine. Accordingly, the target spoke or adaptor performs independent of the output mode, hereinafter referred to as a modeless spoke or adaptor. ~~Advtangeously~~ Advantageously, the output ~~more~~ mode may be more dynamic because the specification of the object and data operation is deferred to the point where an event handler executes an output action. For example, in embodiments with modeless target adaptors, any sequence of data manipulation operations over any set of target objects may be performed within a single step transformation, rather than one output mode at a time. Further, the same record layout may be used to

output to multiple objects. Similarly, it is also possible to define multiple record layouts for use with a single target object. Each record layout can have different mapping expressions and represent different views of the target object. Sharing a record layout between multiple objects can reduce the complexity of a map and using multiple record layouts per object provides greater flexibility in expressing transformation rules. In one particular embodiment, the modeless target adaptors for SQL databases can be configured to write the generated query statements to a script file and/or to execute the query statements as they are generated.

Please replace the paragraph beginning on page 20, line 13 with the following rewritten paragraph:

Field object **366** manages the data and metadata for a single field in a record layout. This may include information about the data type of the field, default value, key markers, documentation, and mapping expressions. Fields ~~object 346~~ 364 is an ordered collection of Field objects **366**, whereby the collection may be used to define a record layout.

Please replace the paragraph beginning on page 20, line 21 with the following rewritten paragraph:

RecordLayout object 362 is a named collection of Fields ~~346~~364. It provides support for storing record layout metadata (name, description, *etc.*) and provides operations for managing the collections of fields at design time and for evaluating mapping expressions during transformation run time. RecordLayouts ~~object 360~~ is used to manage the collection of ~~records~~ record layouts for source **324** or target object **334**. RecordLayouts ~~object 360~~ may also serve to associate the record layouts with the rules used to recognize the records.

Please replace the paragraph beginning on page 20, line 28 with the following rewritten paragraph:

RecordRecognitionRule object **372** may be used to define an association between a specific record layout and a logical condition. RecordRecognitionRules **370** ~~object~~ is the collection of RecordRecognitionRule objects **372**.

Please replace the paragraph beginning on page 21, line 1 with the following rewritten paragraph:

Source object **324** manages the metadata for a transformation source and provides indirect access to the source data object through a source spoke **120**. Sources ~~object 320~~ is the collection of Source objects **324** used for a transformation map.

Please replace the paragraph beginning on page 21, line 4 with the following rewritten paragraph:

Similarly, the Target object **334** manages the metadata for a transformation target and provides indirect access to the target data object through a target spoke **130**. The Targets ~~object~~ **330** is the collection of Target objects **334** used for a transformation map.

A12

Please replace the paragraph beginning on page 21, line 24 with the following rewritten paragraph:

After the data source is located, the source type must be specified. For example, in some embodiments, the user may only have to type in the file name for a delimited ASCII file, or enter table name and passwords if a SQL database is the source. In other embodiments, the user can visually parse records (including binary data) in the case of fixed length ISAM or sequential files. Alternatively, the user can use a dictionary browser to define the source record layout.

A13

Please replace the paragraph beginning on page 23, line 4 with the following rewritten paragraph:

If the source type is delimited ASCII, the present invention may present the user with a hex browser to determine separators, delimiters and starting offset. Advantageously, the user can use these values to search for the source properties. The user can also determine if there is a header record. A header record contains information, such as column headings, but is not actual data. In addition, by scrolling through the data, the user can view several records. After scrolling through the first few records, the user can usually determine what the separators and delimiters are. Most field separators are printable characters such as a comma (,) or a pipe (|). The user will see printable characters in the single line of data. Some field and record separators are non-printable characters such as a Carriage Return-Line Feed or a Tab. A non-printable character will appear as a period (.) in the line of data, but in the line of hex values above the ruler, the user will see the hex value of that character.

A14

Please replace the paragraph beginning on page 23, line 16 with the following rewritten paragraph:

Flat, Fixed ASCII, Binary, and record manager data sources may not be automatically broken into records and/or fields because they do not contain delimiters or separators to mark the locations of field and record breaks. The user must define how the data will be broken up (also called parsed), using the source properties, source record Layout, and source record Parser windows. One way of defining the data structures is through the use of a parsing interface that allows the user to manually parse the data, *e.g.,* by defining record length and starting offset, field sizes, field names, data types, and data properties.

A15

Optionally, the parsing interface may include a data browser that parses source data into the data structures defined by the user, allowing the user to verify that the structure is properly defined.

Please replace the paragraph beginning on page 24, line 7 with the following rewritten paragraph:

In embodiments utilizing modal target adaptors, the present invention allows a great deal of flexibility in defining how data is written to the target by allowing the user to specify an output mode. In one embodiment, the output modes may be:

- **Replace File/Table** – create a new data target or table, OR overwrite both the data and the structure of an existing file or table.
- **Append to File/Table** – keep the existing records and add new records to an existing file or table, OR add records to an empty table.
- **Update File/Table** – searches an existing data target for a match in the key fields, which the user defines in target keys/indexes, and updates data in the specified manner. May be particularly useful when the user has selected a dBASE, ODBC, or SQL target type.
- **Clear File/Table contents and Append** – preserves the target record layout and the relationships between tables (if the user has defined them) and discards any existing records. May be particularly useful if the user has selected an ODBC or SQL target type.
- **Delete from File/Table** – searches an existing data target for a match in the key fields, which the user ~~has~~ defines in target keys/indexes, and deletes data in the specified manner. May be particularly useful when the user has selected an ODBC, or SQL target type~~;~~.

Please replace the paragraph beginning on page 25, line 9 with the following rewritten paragraph:

In one embodiment, the user may use a visual mapping interface to drag and drop and to match fields as the user likes (*e.g.* targetField = sourceField). However, any technique for specifying ~~that~~ where source data should appear in the data target is within the scope of the invention. Further, if the user wants to relate source and target fields, the user can create numeric and logical expressions to massage data and fields into the exact output format the user requires. Still further, the user can extract records that meet a logical condition or fall into a range so that only a subset of the total records passing through the transformation engine are written to the data target. In one embodiment, the transformation system may

default to a one-to-one mapping between source fields and target fields unless the user specifies otherwise, as this is one of the most frequently used mappings. However, any mapping may be used as the default without departing from the scope of the invention.

Please replace the paragraph beginning on page 28, line 12 with the following rewritten paragraph:

Generally, calculations cannot be performed on data that contains non-numeric characters, or is defined as text data type. However, functions may be available that allow the user to convert data from text to numeric values. Examples of functions that perform calculations follow:

- To add the contents of two or more source fields with source fields of "Total" and "Tax" and a target field of "Grand Total":

```
GRAND TOTAL = [Total] + [Tax]
```

- To multiply the contents of two fields from the data source with source field of "Quantity" and "Price" and a target field of "TOTAL SALE":

```
TOTAL SALE = [Quantity] * [Price]
```

- To multiply the contents of one field from the data source by a literal value (add 7.25% sales tax) with a source field of "Total" and a target field of "GRAND TOTAL":

```
GRAND TOTAL = [Total] * 1.0725
```

- To divide the contents of one field by the contents of another field from the data source with source fields of "Total Sale" and "Quantity" and a target field of "Price Per":

```
PRICE PER      = [Total Sale] / [Quantity]
```

It is noted that the preceding examples (along with all of the examples presented herein) are for illustrative purposes only and do not serve to limit the scope of the invention.

Please replace the paragraph beginning on page 38, line 19 with the following rewritten paragraph:

Further, in some embodiments, the user can use a decision structure to define groups of statements that may or may not be executed, depending on the value of an expression. In these embodiments the transformation engine may ~~supports~~ support the following: **For...Next, If...Then...Else, Select Case, While...Wend**. In addition the transformation engine may also support the following error-trapping statements: **On Error, Goto, Resume, Resume Next, Return**.

Please replace the paragraph beginning on page 38, line 25 with the following rewritten paragraph:

The user may use expressions to perform a variety of specialized data manipulation or record filtering. For example, in one embodiment, with a single generic ClearMapPut action, the transformation engine defaults to converting all records in the data source to the data target. However, in this and other embodiments, the user may specify one or more replacement or additional transformation filters. Typical uses for transformation filters include, without limitation, converting records based on a condition, converting a range of records, and converting a random sampling of records. When a transformation is executed (described in detail in conjunction with **FIG. 5**), the transformation engine may first filter records, and ~~the~~ then perform target field expressions to modify the actual data in a particular field. In another embodiment, the user can specify a range on either the source side or the target side of the process. For example, if the user wants the first 100 records from the source and then to extract a subset of those 100 records, the user would set a range in source sample and also set up the record filtering expressions in the target. Conversely, if the user wants the transformation engine to read all the records that meet the record filter expression criteria FIRST, then specify a range of 100 of those records, the user would set the extract logic in the source Filter and the Range in target Filter.

Please replace the paragraph beginning on page 40, line 23 with the following rewritten paragraph:

Other embodiments may include functionality for overflow handling. An overflow occurs when numeric fields in the source have a higher precision than the target numeric fields. The last few digits may be lost or the numeric value may be completely altered. The user may specify rules to handle an overflow occurrence. For example, the user may choose to ignore the overflow, and the transformation will proceed as if the overflow did not occur. The user may choose to treat an overflow occurrence as a warning, and the transformation engine will display a warning message and/or write a message to a log file indicating that an overflow has occurred, but the transformation will continue uninterrupted. Alternatively, the user may choose to treat overflow conditions as an error, whereby the transformation engine will display an error message informing the user that an overflow has occurred and if the maximum error count has been reached, the transformation will be aborted.

Please replace the paragraph beginning on page 42, line 3 with the following rewritten paragraph:

Events are opportunities that can be exploited within the transformation cycle. Put another way, they are moments in the timeline of the transformation. For example, if the user ~~wish~~ wishes something to happen as soon as a record is read into the transformation, the user would choose the **AfterNextrecord** event, and then choose the action the user ~~wish~~ wishes to happen at that time.

Please replace the paragraph beginning on page 42, line 8 with the following rewritten paragraph:

An advantage of the event handling in the present invention is that it provides for far more complex transformations, *e.g.*, with multiple record types on both source and target, than prior art techniques. Further, the present invention allows these complex transformations to be accomplished with very little difficulty. Examples of complex transformations that may be performed by the present invention include, but are not limited to, record aggregation, unrolling of data, transposing of data, and restructuring. The event handling allows the user much of the flexibility, and customizability that the user would get from a custom coded solution, without the hassle of building a custom program every time the user ~~wish~~ wishes to convert data.

Please replace the paragraph beginning on page 42, line 17 with the following rewritten paragraph:

Events may generally be separated into four types: generic source events, generic target events, transformation ~~Events~~ events, and specific type events. The generic source and target events are those that may be triggered during the reading and writing from all data sources and data targets, respectively. In contrast, a specific type event is one that is triggered with respect to a particular data source or target, or a specific type of data structure. Transformation events are those that may be triggered during the actual transformation process.

Please replace the paragraph beginning on page 46, line 22 with the following rewritten paragraph:

Generally, event actions may be processed in a predetermined order, which is dependent on the triggering event. If multiple actions are associated with a single event handler, the actions may be executed in the order in which they are defined. In some embodiments, the user may modify the order of execution.

Please replace the paragraph beginning on page 46, line 26 with the following rewritten paragraph:

In addition to being flexible, the event handling interface is preferably designed to be easy to use. The user chooses the event that the user wishes to have trigger an action, and then ~~choose~~ chooses the action and ~~define~~ defines its parameters. Screen interfaces may be used to help the user define the parameters of each action as it is chosen. The user can choose to have more than one action occur during a particular event, and the user can choose to have the same action occur during more than one event. The present invention imparts no restrictions on the number or type of event actions or triggering events that may be used.

Please replace the paragraph beginning on page 47, line 7 with the following rewritten paragraph:

Further, in some embodiments, either the user or the transformation engine may generate metadata regarding a transformation. In these embodiments, this metadata may be stored with the transformation and/or recorded in any logs that generated. One example of metadata is the version information, which allows the user to specify portable transformation specifications and their associated revision numbers. Thus, when the user revises a transformation specification and ~~want~~ wants to make sure that its date and type are traceable, the user can specify both major and minor transformation revisions. For example a major revision may be numbered in whole numbers, e.g., 1, 2, 3, etc., and a minor revision may numbered with decimals, e.g., 1.1, 1.2, etc. This information may allow the user to keep designed transformations synchronized with transformations being executed by the transformation engine.

Please replace the paragraph beginning on page 47, line 30 with the following rewritten paragraph:

Once the transformation initialization is started **610**, it establishes connections **620** and **630** to transformation sources and transformation targets, respectively. The embodiment shown in **FIG. 6** makes use of a primary target and a reject target. Reject targets may be used to ~~stored~~ store records that do not pass a filter or otherwise are not written to a primary target. Reject targets are often useful in debugging and data analysis. Accordingly, the next inquiries **640** and **644** are whether both of the targets (*i.e.,* the primary and reject) have record layouts specified for them. The illustrated embodiment is configured to use (**642** and **646**) the source layout for the targets, if none has been specified. In other embodiments, different default actions may be used, including, without limitation, requiring that the user select or specify a layout, and simply generating an error event. After establishing record layouts, the data targets are opened in preparation to receive output **650**.

Please replace the paragraph beginning on page 48, line 29 with the following rewritten paragraph:

If there are no unhandled exceptions, the system may fire a **BeforeNextRecord** event **740** and retrieves the next source record **750**. By iterating through the data source(s) on a record-by-record basis, the whole data source does not have to be read into memory. Further, this also allows for easy navigation of hierarchical data structures, as the present invention can keep track of ~~interrecord~~ relationships between records. As the transformation process of the present invention iterates over records from the data source, the last instance of each distinct record type may be remembered. Thus, in instances where data is either hierarchical or there is an implied parent-child relationship, mapping expressions may address the data from any of the ancestors from the last record. Though records are a commonly used data structure, particularly with database sources, any size or type data structure may be read in step **750**.

Please replace the paragraph beginning on page 49, line 9 with the following rewritten paragraph:

Next, if the end of the source file is detected **760**, an EOF event may be fired **762**. If no other data source files remain then the transformation loop is ended **790**, otherwise the loop proceeds to step **730**.

Please replace the paragraph beginning on page 49, line 12 with the following rewritten paragraph:

If the end of the source file is not detected 760, retrieved source record **750** is checked for input error **764**. If input error is detected, it is handled **766** by one or more error handlers. An error handler is similar to an event handler in that it is designed to detect occurrence of an event (*i.e.*, an error condition) and execute the appropriate action. The appropriate action may be specified by the user or may be predefined. If the input error is handled then the loop moves on to the same step (**770**) as if the input error did not occur. If the input error is not handled **768**, the transformation loop may be ended **790** unsuccessfully.

Please replace the paragraph beginning on page 49, line 19 with the following rewritten paragraph:

If the input error is handled successfully **768** or if there is no input error **764**, data ~~change~~ changes are ~~evaluate~~ evaluated by monitors **770** to determine if any data monitors are triggered **772** (*see* **FIG. 8**). This allows for **OnAnyDataChange** events to be triggered **742**. If all of the active data monitors are triggered, an **OnAllDataChange** event may be fired **744**. If not, a sub-loop (**780, 782**, and **788**) may be executed that checks whether each individual ~~the~~ data monitor has been triggered ~~784~~ 784, and if so, fires

the appropriate **OnDataChange** event **786**. After the sub-loop has executed, a record-layout-specific **AfterNextRecord** event **746** may be fired, followed by a generic **AfterNextRecord** event **748**.

Please replace the paragraph beginning on page 56, line 10 with the following rewritten paragraph:

Transformation projects created through the project designer interface may support several types of project steps:

- **Start**. This is a special step that represents the starting point for a transformation project. It may also be used for the definition and initialization of global project variables used to pass information between different project steps.
- **Transformations**. Users may be able to create new transformation steps or use existing transformation specifications. Transformation steps may be able to share session information with other transformation steps to simplify the specification of login information and to facilitate the coordination of transactions across the entire project.
- **Decision**. The decision step provides a way for project to conditionally control the project flow using an if-then-else logic. Decision steps ~~supports~~ support the use of simple expressions to test for branch conditions. Expressions may have access to information stored, global variables, and completion code information from prior steps to determine which branch of the process flow continues execution.
- **Expression**. Expressions can be evaluated to compute values or to execute functions from an external library. Expressions have access to global project variables.
- **Application**. The application step allows users to incorporate the execution of external applications such as schedulers, electronic mailers, sort utilities, bulk loaders, and indexing utilities.
- **SQL Statement**. The SQL statement steps may provide an effective mechanism for users to execute SQL DDL for the SQL resources used in a project. Users may be able to define tables, indexes, primary keys, and relational constraints. The combination of the SQL statement steps and the transformation steps make it possible for the transformation engine to populate entire databases at one time.
- **Sub-Project**. The use of sub-projects may allow smaller projects to be combined into bigger projects and provides a mechanism for project reuse. Sub-projects may also

A33

serve to give the user greater control over how transactions are coordinated between multiple transactional steps.

- **Stop**. The stop step represents the end of a project. It can be used to specify project termination expressions.

Please replace the paragraph beginning on page 67, line 2, with the following rewritten paragraph:

A system and method is described for event-driven data transformation. Generally, the system and method is directed to a transformation engine that iterates through one or more data sources, transforms data ~~receive~~ received from the data sources, and stores the output to one or more data targets. More specifically, the transformation engine is driven by executing specified event actions upon occurrence of specified triggering events. Thus, flexible, adaptable, highly tailored transformations can be implemented without incurring the often substantial expense of developing customized point-to-point solutions from scratch. The present invention supports one-to-one mappings, many-to-one mappings, one-to-many mappings, and many-to-many mappings. In addition, the present invention supports both hierarchical and flat data sources and targets.

A34